Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Рязанский государственный радиотехнический университет

имени В. Ф. Уткина»

Кафедра «Вычислительная и прикладная математика»

**Отчет**

**по лабораторной работе № 5**

**по дисциплине**

**«Объектно-ориентированное программирование»**

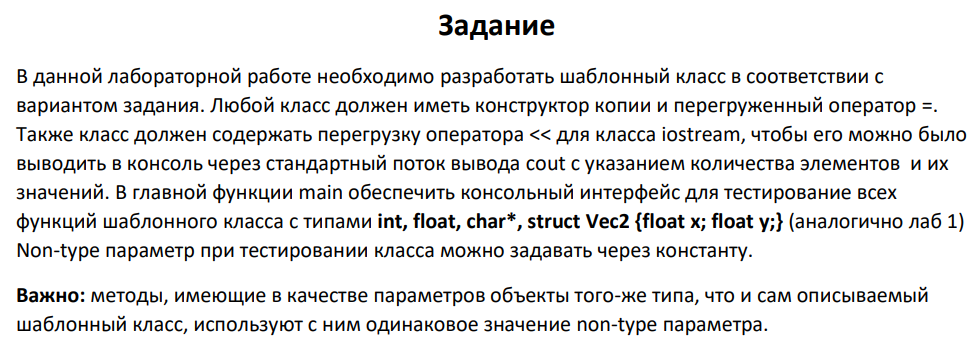
**на тему**

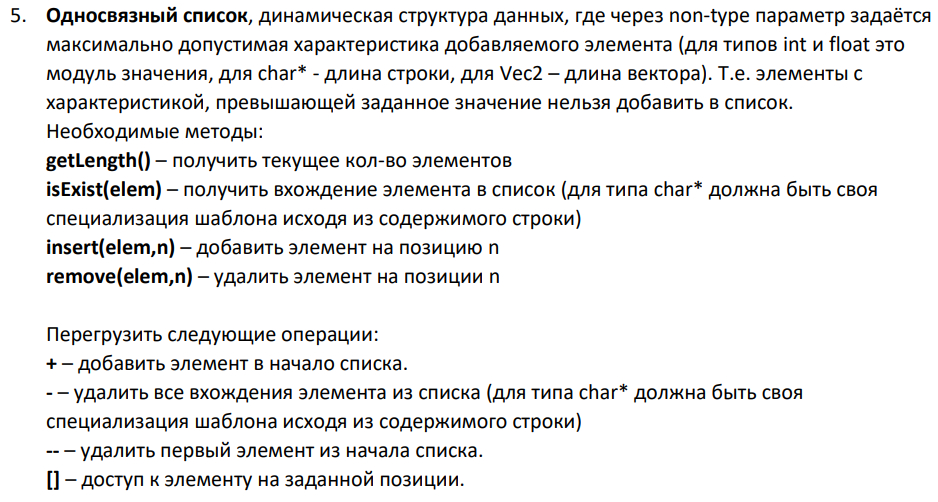
**«Шаблонные классы»**

Выполнил:   
студент гр. 143  
Вербицкая И. С.

Проверил:  
Антипов О.В.

Рязань 2022





**Описание структуры программы**

**Класс Object**

***Поля класса:***

* static int All – количество созданных объектов
* static int Act - количество активных объектов
* int\* Op – указатель на динамический массив операций, выполненных с объектом (каждой операции соответствует свой целочисленный номер)
* int Ops – количество операций в массиве Op

***Методы класса:***

* static void RestartObjInfo() – сбрасывает информацию о количестве объектов класса (созданных и активных)
* Object() – конструктор класса
* Object(const Object& Obj) – конструктор копирования класса
* ~Object() – деструктор класса
* static void PrintTotalInfo() – выводит в консоль информацию о количестве объектов класса (созданных и активных)
* static int GetAll() – геттер поля All
* static int GetAct() – геттер поля Act
* void AddOp(int k) – добавляет команду k в динамический массив операций Op, выделяя под неё память
* void PrintOp() – выводит список операций текущего объекта
* void ClearOp() – сбрасывает список операций над объектом, освобождая выделенную память
* Object& operator= (Object& Matr) – перегруженный оператор присваивания для класса
* bool Comp(int (&Spis)[N]) – вспомогательная функция, возвращающая 1 в случае, если массив операций Op совпадает с массивом Spis

**Класс Matrix3x3** – класс-потомок класса Object

***Поля класса:***

* arr - двумерный массив действительных чисел размерностью 3 на 3
* id - логическое значение; 1 – если матрица единичная; 0 – если нет

***Методы класса:***

* bool Ident () - функция по определению поля id в структуре матрицы; возвращает 0 если матрица не единичная и 1 – если единичная
* Matrix3x3 (float(&Matr)[N][N]) – конструктор класса; заполняет матрицу значениями двумерного массива Matr
* Matrix3x3(const Matrix3x3& Matr) – конструктор копирования класса; копирует значения полей Matr в поля текущего объекта
* ~Matrix3x3() – деструктор класса
* static Matrix3x3 St(int A, int B) – статический метод, возвращающий новый объект – случайным образом сгенерированную матрицу
* Matrix3x3 operator+ (Matrix3x3& A) – возвращает новый объект класса – матрицу – результат сложения текущей матрицы с матрицей А; добавляет операцию сложения в списки операций текущей матрицы и матрицы А
* Matrix3x3 operator\* (Matrix3x3& A) – возвращает новый объект класса – матрицу – результат умножения текущей матрицы и матрицы А; добавляет операцию умножения в списки операций текущей матрицы и матрицы А
* void operator\* (const int& n) – умножает каждый элемент матрицы на скаляр n; добавляет операцию умножения на скаляр в список операций текущей матрицы
* void operator++ () – транспонирует матрицу; добавляет операцию транспонирования в список операций текущей матрицы
* float Det () – находит и возвращает определитель матрицы; добавляет операцию нахождения определителя в список операций текущей матрицы
* void operator! () - преобразует матрицу в обратную, если это возможно; добавляет операцию обращения в список операций текущей матрицы, если обратная матрица для данной существует
* void Out () – выводит матрицу в консоль
* bool Compare (float (&Matr)[N][N]) – сравнивает поля матрицы с матрицей Matr, переданной как двумерный массив
* Matrix3x3& operator= (Matrix3x3 Matr) – копирует значения полей матрицы Matr в поля текущей матрицы и возвращает указатель на текущий объект

**Класс Test**

***Поля класса:***

* Pass – целочисленное значение – количество успешно пройденных тестов
* Numb – целочисленный счётчик количества тестов в целом

***Методы класса:***

* void Start () – обнуляет счётчики в полях класса
* bool Expect (int& Actual, const int& Expected), bool Expect (float& Actual, const float& Expected) – сравнивают значения Actual и Expected между собой и возвращает 1, если они совпадают
* bool ExpectM(Matrix3x3& Actual, int (&Expected)[N]) – сравнивает список операций матрицы Actual со списком операций – одномерным массивом Expected – и возвращает 1, если они совпадают
* bool ExpectM (Matrix3x3& Actual, float (&Expected)[N][N]) сравнивает объект - матрицу Actual – с двумерным массивом - матрицей Expected - и возвращает 1, если они совпадают
* void Success (bool n) – выводит сообщение об успехе/неудаче теста
* void Results () – выводит результаты теста
* void Head() – выводит заголовок теста (его номер)
* void Run () – запускает Unit-тестирование
* void CountOfAct1(), void CountOfAct2() – вспомогательные процедуры для теста CountOfAct(), описанного ниже; инициализируют по одной матрице

*Все нижеописанные методы возвращают 1 в случае совпадения полученного результата с ожидаемым и 0 – в противном случае.*

* bool TSumm () – тестирование метода Summ класса Matrix3x3
* bool TMult () - тестирование метода Mult класса Matrix3x3
* bool TSclr () - тестирование метода Scalar класса Matrix3x3
* bool TTransp () - тестирование метода Transpos класса Matrix3x3
* bool TDet () – тестирование метода Det класса Matrix3x3
* bool TRev () - тестирование метода Reverse класса Matrix3x3
* bool TMath () – тестирование методов Reverse и Mult на основе соблюдения формулы линейной алгебры
* bool TDetNull () - тестирование метода Det класса Matrix3x3 (нахождение определителя пропорциональной матрицы)
* bool TSummTranspSclr () - тестирование методов Summ, Transpos, Scalar класса Matrix3x3
* bool TRevNever () - тестирование метода Reverse класса Matrix3x3 (попытка обращения матрицы с нулевым определителем)
* bool CountOfAll() – тестирование метода St класса Matrix3x3, правильности работы конструкторов копирования для классов Matrix3x3 и Object, правильности подсчёта общего количества созданных объектов (поле All класса Object)
* bool CountOfAct() - тестирование метода St класса Matrix3x3, правильности работы конструкторов копирования для классов Matrix3x3 и Object, правильности подсчёта общего количества активных объектов (поле Act класса Object)
* bool TrueCopy() – тестирование правильности перегрузки оператора присваивания для классов Matrix3x3 и Object
* bool TrueOperations1() – тестирование метода AddOp класса Object путем преобразований заданной матрицы
* bool TrueOperations2() - тестирование правильности перегрузки оператора присваивания для классов Matrix3x3 и Object и метода AddOp класса Object

**Листинг программы:**

*Файл BaseList.h*

#pragma once

#include <iostream>

#include <math.h>

#include <stddef.h>

using namespace std;

//структура узла односвязного списка

template <typename T>

struct Node

{

T Val; //содержимое элемента

Node\* Next; //указатель на следующий узел

};

//односвязный динамический список (общий шаблон)

template <class T, int MaxSize>

class BaseList

{

protected:

int Len; //кол-во элементов списка

Node<T>\* Head; //первый элемент списка

bool Empty() const { return Head==nullptr; } //пуст ли список?

virtual bool isCompatible (Node<T>\* Elem) = 0;//подходит ли элемент под ограничение MaxSize?

virtual bool isEqual (T a, T b) { return a==b; }//равны ли элементы?

//удаляет элемент по его адресу

virtual void DelAddr(Node<T>\* Addr) {

if (Addr!=nullptr) {

if (Addr==Head) {

Head=Addr->Next;

delete Addr;

Addr = nullptr;

Len--;

}

else {

Node<T>\* Dop;

Dop=Head;

while (Dop->Next!=Addr)

Dop=Dop->Next;

Dop->Next=Addr->Next;

delete Addr;

Addr = nullptr;

Len--;

};

}

}

//конструтор копирования для большинства типов

/\* virtual void CreateValue (T& Value, T Elem) { Value = Elem; }\*/

public:

BaseList() { Head = nullptr; Len = 0; }

virtual ~BaseList() { Clear(); }

//кол-во элементов в списке

int getLength() const { return Len; }

//не выходит ли индекс за пределы списка?

bool isIndex(int ind) const { return (ind<Len) && (ind>=0); }

//полная очистка списка

void Clear() {

while (!Empty())

DelAddr(Head);

Len = 0;

}

//проверка вхождения элемента в список

bool isExist(T Elem) {

bool F = false;

Node<T>\* Dop;

Dop=Head;

while (Dop!=nullptr) {

//if (Dop->Val==Elem)

if (isEqual(Dop->Val,Elem))

F = true;

Dop = Dop->Next;

}

return F;

}

//вывод содержимого списка

virtual void print () {

cout << "\n| ОДНОСВЯЗНЫЙ СПИСОК:" << endl;

Node<T>\* Dop;

Dop=Head;

for(int i = 0; i < Len; ++i) {

cout << "| "<< i+1 << " | " << Dop->Val << "\n";

Dop=Dop->Next;

}

cout << "| КОЛ-ВО ЭЛЕМЕНТОВ: " << Len << endl;

cout << "| МАКС ХАРАК-ТИКА: " << MaxSize << endl;

}

//удалить все вхождения элемента из списка

T& operator- (T Elem) {

while (isExist(Elem))

for (int i=0; i<Len; i++)

remove(Elem,i);

}

//удалить первый элемент из начала списка

T& operator-- () { if (!Empty()) remove(Head->Val,0); }

//добавить элемент в начало списка

T& operator+ (T Elem) { insert(Elem,0); }

//добавить элемент на позицию n

void insert(T Elem, int n) {

Node<T>\* Dop;

Dop=Head;

Node<T>\* New = new Node<T>; //выделяем память под узел

CreateValue(New->Val,Elem);

//New->Val = Elem;

Len++;

if (isCompatible(New)) {

if (Empty()) { //вставка в пустой список

New->Next=nullptr;

Head = New;

}

else if (n==Len-1) { //вставка в конец списка

while ((Dop->Next)!=nullptr) Dop=Dop->Next;

New->Next=nullptr;

Dop->Next=New;

}

else if (n==0) { //вставка в начало списка

New->Next = Head;

Head = New;

}

else if ((n<Len-1)&&(n>0)) { //вставка в середину списка

int i=0;

while (i<n-1) { //приходим к предшествующему n элементу

Dop=Dop->Next;

i++;

}

New->Next = Dop->Next; //Наш новый эл указывает на эл, идущие после n-го

Dop->Next = New; //Эл перед n-ым начинает указывать на новый эл

}

else { //попытка вставки на недопустимый индекс

delete New;

New=nullptr;

Len--;

}

}

else { //попытка вставки недопустимого по MaxSize значения

delete New;

New=nullptr;

Len--;

}

}

//конструтор копирования для большинства типов

virtual void CreateValue (T& Value, T Elem) { Value = Elem; }

//удалить элемент с позиции n

void remove(T Elem, int n) {

if (isExist(Elem)) {

int D = 0;

Node<T>\* Dop;

Dop=Head;

while (Dop!=nullptr) {

//if ((D==n)&&(Dop->Val==Elem)) {

if ((D==n)&&(isEqual(Dop->Val,Elem))) {

DelAddr(Dop);

break;

}

D++;

Dop = Dop->Next;

}

}

}

//доступ к элементу на заданной позиции

T& operator[](int index) {

if ((index<Len) && (index>=0)) {

Node<T>\* Dop;

Dop=Head;

for (int i=0;i<index;i++)

Dop=Dop->Next;

return Dop->Val;

}

}

};

*Файл List.h*

#include "BaseList.h"

#include <strings.h>

using namespace std;

//--------------------- int, float -------------------------

template <class T, int MaxSize>

class List: public BaseList<T,MaxSize>

{

//подходит ли элемент под ограничение MaxSize?

bool isCompatible (Node<T>\* Elem) override {

return fabs(Elem->Val)<=static\_cast<float>(MaxSize);

}

};

//------------------------ Vec2 ----------------------------

struct Vec2 {

int x;

int y;

//конструктор копирования

Vec2& operator= (Vec2 Value) {

x = Value.x;

y = Value.y;

return \*this;

}

bool operator== (Vec2 Value) {

return (x == Value.x) && (y == Value.y);

}

bool operator!= (Vec2 Value) {

return (x != Value.x) || (y != Value.y);

}

};

// перегрузка оператора << класса ostream для Vec2

// для вывода в стандартном потоке вывода

ostream& operator<<(ostream& os, const Vec2& vec) {

return os << "(" << vec.x << "; " << vec.y << ")";

}

template <int MaxSize>

class List<Vec2,MaxSize>: public BaseList<Vec2,MaxSize>

{

//подходит ли элемент под ограничение MaxSize?

bool isCompatible (Node<Vec2>\* Elem) override {

int X = Elem->Val.x;

int Y = Elem->Val.y;

int R = MaxSize;

return X\*X+Y\*Y<=R\*R;

}

};

//----------------------- char \* ---------------------------

template <int MaxSize>

class List<char\*,MaxSize>: public BaseList<char\*,MaxSize>

{

public:

~List() { ClearContent(); }

private:

//для всех элементов списка удаляет память,

//выделенную под строки, хранимые в эл-тах

void ClearContent() {

if (!this->Empty()) {

Node<char\*>\* Dop;

Dop = this->Head;

while (Dop!=nullptr) {

delete Dop->Val;

Dop->Val = nullptr;

Dop = Dop->Next;

}

}

}

//равны ли элементы?

bool isCompatible (Node<char\*>\* Elem) override { return strlen(Elem->Val) < MaxSize; }

bool isEqual (char\* a, char\* b) override { return strcmp(a,b)==0; }

//bool operator== (char\* a, char\* b) {return strcmp(a,b)==0}

//конструктор копии

void CreateValue (char\*& Value, char\* Elem) override {

Value = new char[strlen(Elem)];

if(Elem) strcpy(Value, Elem);

else strcpy(Value, " ");

}

/\*char\* operator= (const char\* value) {

printf("OPA COPY");

char\* S = new char[strlen(value)];

if(value) strcpy(S, value);

else strcpy(S, " ");

return S;

}\*/

void DelAddr(Node<char\*>\* Addr) override {

//cout << "Удаляем " << Addr->Val << endl;

delete Addr->Val;

Addr->Val = nullptr;

BaseList<char\*,MaxSize>::DelAddr(Addr);

}

};

*Файл Menu.h*

#pragma once

#include <iostream>

#include "List.h"

#define MAXSTR 256

const int MAXSIZE = 16;

using namespace std;

//Предобъявления

void MainMenu();

void CreateMenu(int Key);

template <class T, int MaxSize>

void ListMenu(List<T,MaxSize>& MyList);

template <class T, int MaxSize>

void InputElementMenu(List<T,MaxSize>& MyList, int Key);

//Стартовое меню

void MainMenu() {

int Key; //номер вводимой команды

while (true ){

system("cls");

cout << "\n----- СОЗДАНИЕ ШАБЛОННОГО СПИСКА -----\n" << endl;

cout << " Список какого типа будем тестировать?" << endl;

cout << "| 1 | int |" << endl;

cout << "| 2 | float |" << endl;

cout << "| 3 | Vec2 |" << endl;

cout << "| 4 | char\* |" << endl;

cout << "| 5 | Никакого! (Завершить работу) |" << endl;

cout << "\nВведите номер команды: "; cin >> Key;

if ((Key>=1)&&(Key<=4))

CreateMenu(Key);

else if (Key==5)

exit(0);

else {

puts("Такой команды не предусмотрено!");

system("pause");

}

}

}

//Создание списка

void CreateMenu(int Key) {

switch (Key) {

case 1: { List<int, MAXSIZE> L; ListMenu(L); break; }

case 2: { List<float,MAXSIZE> L; ListMenu(L); break; }

case 3: { List<Vec2, MAXSIZE> L; ListMenu(L); break; }

case 4: { List<char\*,MAXSIZE> L; ListMenu(L); break; }

}

}

//Работа с шаблонным списком

template <class T, int MaxSize>

void ListMenu(List<T,MaxSize>& MyList) {

int n;

int Key;

while (true) {

system("cls");

cout << "\n-------------------- РАБОТА С ШАБЛОННЫМ СПИСКОМ --------------------\n" << endl;

cout << "Какую команду вы хотите выполнить со списком?" << endl;

cout << "| 1 | getLength() | узнать текущее количество элементов |" << endl;

cout << "| 2 | isExist(elem)) | узнать, есть ли элемент в списке |" << endl;

cout << "| 3 | insert(elem,n) | добавить элемент elem на позицию n |" << endl;

cout << "| 4 | remove(elem,n) | удалить элемент elem с позиции n |" << endl;

cout << "| 5 | + | добавить какой-либо элемент в начало списка |" << endl;

cout << "| 6 | - | удалить все вхождения какого-то элемента |" << endl;

cout << "| 7 | -- | удалить первый элемент из начала списка |" << endl;

cout << "| 8 | [] | узнать значени элемента по его индексу |" << endl;

cout << "| 9 | Вернуться к меню создания списка |" << endl;

MyList.print();

cout << "\nВведите номер команды: "; cin >> Key;

switch (Key) {

case 1: { printf("\nКол-во элементов в списке: %d\n", MyList.getLength()); break; }

//case (2,3,4,5,6): case (2-6):

case 2: case 3: case 4: case 5: case 6:

{ InputElementMenu(MyList,Key); break; }

case 7: { --MyList; MyList.print(); break; }

case 8: {

cout << "n = "; cin >> n;

if (MyList.isIndex(n-1)) cout << "\nЭлемент № "<< n << ": " << MyList[n-1] << endl;

else cout << "\nИндекс выходит за пределы списка!" << endl;

break;

}

case 9: { MyList.Clear(); MainMenu(); break; }

default:{ puts("Такой команды не предусмотрено!"); }

}

printf("\n"); system("pause");

}

}

//Работа с шаблонным списком + ввод элемента (специализации в зависимости от типа элемента)

// общая реализация

template <class T, int MaxSize>

void InputElementMenu(List<T,MaxSize>& MyList, int Key) {

//ввод элемента и его индекса (при необходимости)

T elem;

cout << "\nelem = "; cin >> elem;

int n = 0;

if ((Key == 3) || (Key == 4)) { cout << "n = "; cin >> n; }

switch (Key) {

case 2: {

if (MyList.isExist(elem)) puts("Да, такой элемент есть в списке");

else puts("Нет, такого элемента в списке нет");

break; }

case 3: { MyList.insert(elem,n-1); MyList.print(); break; }

case 4: { MyList.remove(elem,n-1); MyList.print(); break; }

case 5: { MyList+elem; MyList.print(); break; }

case 6: { MyList-elem; MyList.print(); break;}

}

}

//Реализация под ввод Vec2

template <int MaxSize>

void InputElementMenu(List<Vec2,MaxSize>& MyList, int Key) {

//ввод элемента и его индекса (при необходимости)

int x; int y; Vec2 elem;

cout << "\nelem:\n";

cout << "x = "; cin >> x;

cout << "y = "; cin >> y;

elem = {x,y};

int n = 0;

if ((Key == 3) || (Key == 4)) { cout << "n = "; cin >> n; }

switch (Key) {

case 2: {

if (MyList.isExist(elem)) puts("Да, такой элемент есть в списке");

else puts("Нет, такого элемента в списке нет");

break; }

case 3: { MyList.insert(elem,n-1); MyList.print(); break; }

case 4: { MyList.remove(elem,n-1); MyList.print(); break; }

case 5: { MyList+elem; MyList.print(); break; }

case 6: { MyList-elem; MyList.print(); break;}

}

}

//Реализация под ввод char\*

template <int MaxSize>

void InputElementMenu(List<char\*,MaxSize>& MyList, int Key) {

//ввод элемента и его индекса (при необходимости)

char\* s = new char[MAXSTR];

cout << "\nelem = "; cin >> s;

int n = 0;

if ((Key == 3) || (Key == 4)) { cout << "n = "; cin >> n; }

switch (Key) {

case 2: {

if (MyList.isExist(s)) puts("Да, такой элемент есть в списке");

else puts("Нет, такого элемента в списке нет");

break; }

case 3: { MyList.insert(s,n-1); MyList.print(); break; }

case 4: { MyList.remove(s,n-1); MyList.print(); break; }

case 5: { MyList+s; MyList.print(); break; }

case 6: { MyList-s; MyList.print(); break;}

}

delete s; s = nullptr;

}

*Файл main.cpp*

#include "Menu.h"

int main() {

system("chcp 1251");

MainMenu();

//работа со строками напрямую

/\* List<char\*,30> L;

char\* s1 = new char[128];

strcpy(s1, "Стр1");

char\* s2 = new char[128];

strcpy(s2, "Стр2");

char\* s3 = new char[128];

strcpy(s3, "Стр3");

L.insert(s1,0);

L.insert(s2,0);

L.insert(s3,2);

L.insert("mda",3);

L.insert("mda",4);

L.print();

L.remove(s2,0);

L+"teststr";

L-"mda";

--L;

delete[] s1; // Удаляем строки!

delete[] s2;

delete[] s3;

L.print();

\*/

//работа с int

/\* List<int,6> MyList;

MyList.insert(2,0);

MyList.insert(3,1);

MyList.insert(4,2);

MyList.insert(66,2);

MyList.insert(2,5);

MyList.print();

MyList.insert(5,3);

MyList.remove(5,3);

MyList+6;

--MyList;

MyList-2;

MyList.print();

\*/

//работа с Vec2

/\* List<Vec2,4> L;

L.insert({2,3},0);

L.insert({2,1},1);

L.insert({2,2},2);

L.print();

Vec2 V = {1,1};

L+V;

L+V;

L-V;

--L;

L.print();

\*/

return 0;

}

**Результаты работы программы:**

Итак, после запуска программа вывела в консоль результаты всех 15-ти тестов (рисунки 1, 2, 3 – приведены результаты выполнения последних 5ти тестов, добавленных в этой лабораторной работе – остальные тесты не изменились), значения которых соответствуют действительности и совпадают с ранее выполненными проверками. Программа работает корректно.

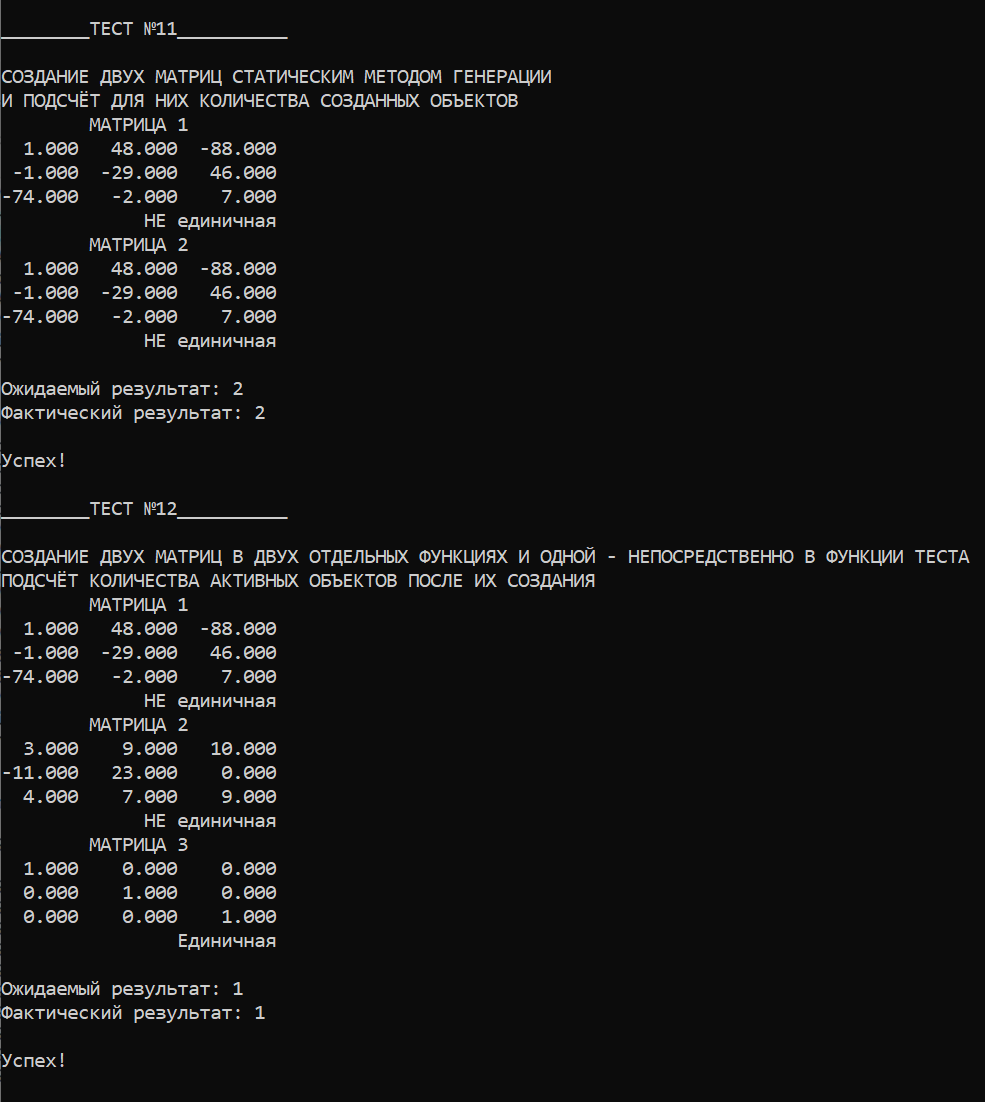


Рисунок 1

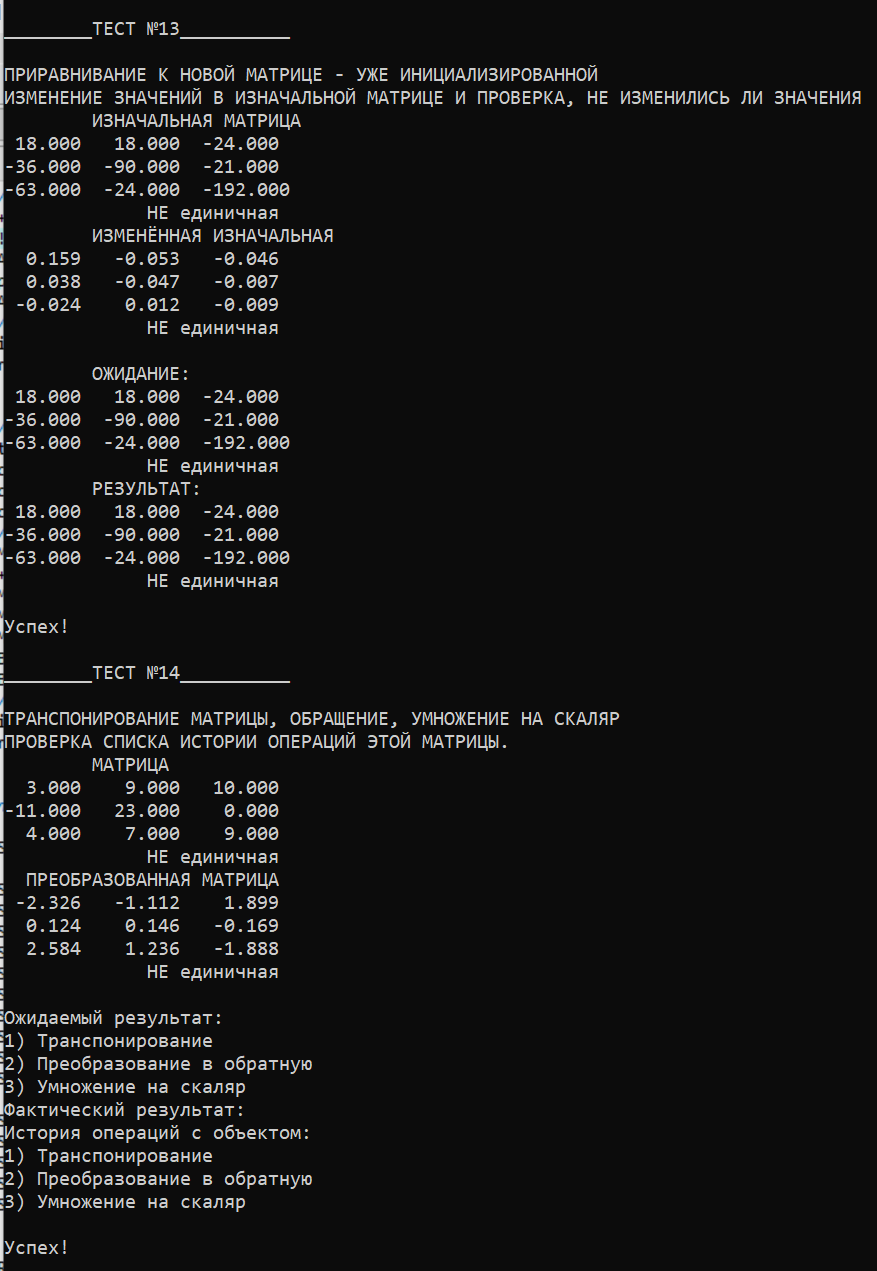


Рисунок 2

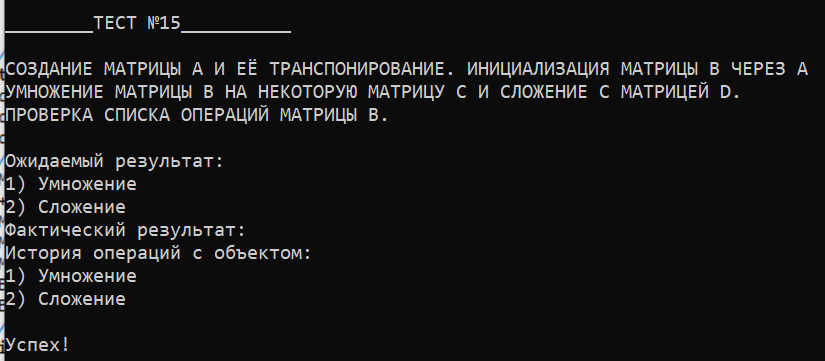


Рисунок 3